Санкт-Петербургский государственный университет

Кафедра механики управляемого движения

Загайнов Илья Андреевич

Выпускная квалификационная работа

Роевой алгоритм в задачах группового управления подвижными объектами

Направление 010400

Прикладная математика, фундаментальная информатика и программирование

Научный руководитель

к.ф.-м.н.

доцент

Алферов Г. В.

Санкт-Петербург

2017

Содержание

Глава 1. Введение

1.1. Стратегии управления группой роботов

Как правило, в задачах робототехники рассматриваются проблемы управления одиночными роботами. Однако не сложно выделить целый ряд задач, где использование группы роботов является более предпочтительным или даже необходимым условием, например, задачи, связанные с логистикой или картографией, в которых использование многоагентных систем существенно упрощает задачу [1].

В связи с этим выделяют отдельную область робототехники, групповую робототехнику, которая рассматривает два различных подхода к решению поставленных перед ней задач: централизованные и децентрализованные стратегии управления.

Первый подход предполагает, что решение о дальнейших действиях роботов принимает внешнее управляющее устройство. Централизованные стратегии очень требовательны к каналу связи и требуют больших вычислительных мощностей, так как управляющее устройство должно обрабатывать информацию, исходящую от всех роботов группы. Еще одним существенным минусом является тот факт, что группа не способна выполнить поставленную перед ней задачу в случае выхода из строя управляющего устройства.

Другой подход лишен упомянутых выше недостатков. В децентрализованных стратегиях управления каждый робот группы обладает

собственным управляющим устройством. Алгоритмы для подобных стратегий могут быть построены таким образом, что роботу не нужно знать информацию обо всех агентах группы, что уменьшает нагрузку на каналы связи и вычислительные устройства отдельных роботов. Более того, выход из строя отдельного робота не сказывается на работоспособности других членов группы. Таким образом, децентрализованные стратегии управления представляют больший интерес для изучения.

Масштабируемость группы является одним из ключевых критериев в групповых стратегиях управления. Увеличение численности группы влечет за собой увеличение затрат на производство роботов. Однако стоимость группы можно уменьшить путем упрощения отдельных агентов. Поэтому поведение таких простых организмов, как насекомые, со временем нашло свое применение в робототехнике. Несмотря на примитивность отдельных взаимодействий, группа насекомых способна решать относительно сложные задачи, стоящие перед ней.

Наиболее интересные стратегии управления зачастую появляются в следствии наблюдения за социальными насекомыми [2, 3]. К таким насекомым относятся, например, пчелы, которым и обязано название одного из наиболее известных подходов к групповому управлению. Речь идет о роевом интеллекте.

Этот термин стал известен благодаря работам Герардо Бени над клеточными роботами в конце 80-ых годов прошлого столетия [4]. Основным принципом роевого интеллекта является решение сложных задач, стоящих перед группой, посредством простых локальных взаимодействий отдельных агентов. Исключительно локальные взаимодействия и асинхронная работа алгоритма позволяют применять роевой интеллект для групп с любым количеством роботов.

1.2. Формальное определение роя и стоящей перед ним задачи

Введем формальное определение роя, используемое И. А. Каляевым в своих работах [5]. Пусть группа состоит из $\boldsymbol{N}$ одинаковых агентов, которые образуют множество $\boldsymbol{O} = \{\boldsymbol{o_1} \dots \boldsymbol{o_N}\}$. Такая группа будет являться гомогенной. Состояние каждого отдельного робота $\boldsymbol{o_i}$ будет описываться вектором $\boldsymbol{S_i(t)} = \{\boldsymbol{s_{i,1}} \dots \boldsymbol{s_{i,m}}\},\ \boldsymbol{i} = \overline{\boldsymbol{1,N}}$. Кроме того, каждый робот может выполнять одно из действий, описанных вектором $\boldsymbol{A_i} = \{\boldsymbol{a_{i,1}} \dots \boldsymbol{a_{i,m}}\},\ \boldsymbol{i} = \overline{\boldsymbol{1,N}}$. На эти действия в общем случае могут накладываться какие-либо ограничения вида $\boldsymbol{A_i\ \epsilon\ \omega}$, где $\boldsymbol{\omega}$ – множество допустимых действий. Среда, в которой функционирует данная группа описывается вектором $\boldsymbol{E(t)} = \{\boldsymbol{e_1} \dots \boldsymbol{e_w}\}$. Состояние среды может быть неизвестно в начале работы алгоритма, в таком случае принято говорить о недетерминированности среды, информацию о которой роботы должны получить в процессе выполнения алгоритма.

Задачей, стоящей перед роем, является достижение конечного состояния $\boldsymbol{S^b} = \{\boldsymbol{S_1^b} \dots \boldsymbol{S_N^b}\}$, где $\boldsymbol{S_i^b}$ – конечное состояние $\boldsymbol{i}$-ого робота, при котором некоторый функционал $\boldsymbol{Y} = \boldsymbol{F(S, A, E)}$ достигает своего минимума. При такой формулировке решением задачи будет являться последовательность действий $\boldsymbol{A} = \{\boldsymbol{A^1} \dots \boldsymbol{A^M}\}$, где вектор $\boldsymbol{A^1} = \{\boldsymbol{A_1^1} \dots \boldsymbol{A_N^1}\}$ описывает действия, предпринятые роботами на первом шаге алгоритма, $\boldsymbol{A^M}$ – на последнем.

Глава 2. Постановка задачи

Основной задачей данной работы является создание роевого интеллекта для использования в децентрализованной самоорганизующейся группе роботов $\boldsymbol{O} = \{\boldsymbol{o_1} \dots \boldsymbol{o_N}\}$ произвольной численности $\boldsymbol{N}$. Перед данной группой стоит цель максимизации сплошного покрытия некоторой плоскости $\boldsymbol{S}$ в недетерминированной среде при условии возможного наличия препятствий $\boldsymbol{B} = \{\boldsymbol{b_1} \dots \boldsymbol{b_P}\}$ произвольного количества $\boldsymbol{P}$, которые помимо ограничений на местоположение роботов $\boldsymbol{S_i} = \{\boldsymbol{x}, \boldsymbol{y}\}$, $\boldsymbol{i} = \overline{\boldsymbol{1}, \boldsymbol{N}}$, также накладывают ограничения на области видимостей этих роботов.

Под областью видимости стоит понимать некоторую область, которую робот может наблюдать благодаря техническому зрению. Эта область задается окружностью радиуса $\boldsymbol{R}$, центром которой является сам робот. Сплошное покрытие подразумевает отсутствие ненаблюдаемых участков внутри группы роботов.

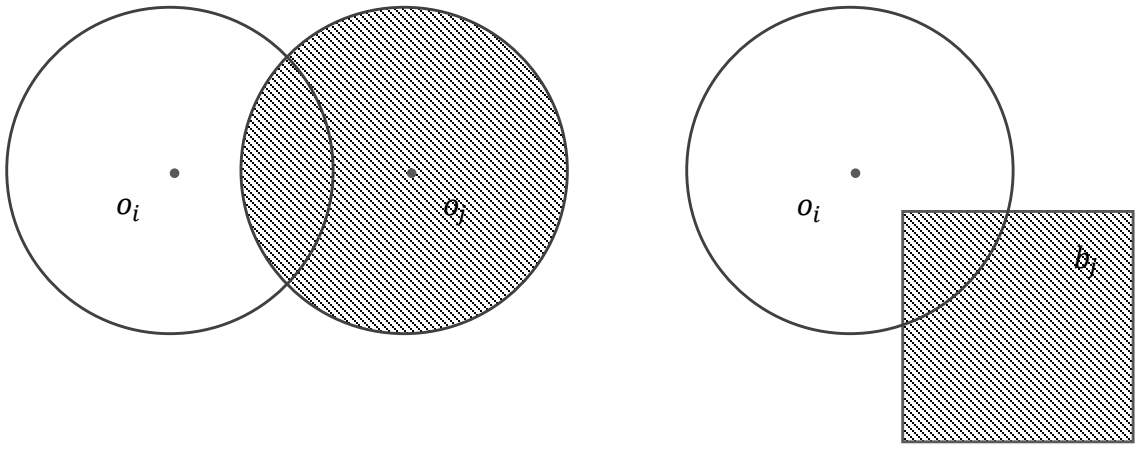

Рис. 1: Возможные варианты пересечения области видимости робота o_i с областью видимости робота o_j и препятствием b_j

Задача может быть сведена к минимизации следующего функционала

$$\boldsymbol{Y = \sum_{i=1}^{N}\left(\sum_{j=i+1}^{N} F_1(o_i, o_j) + \sum_{j=1}^{P} F_2(o_i, b_j)\right)}, \qquad (2.1)$$

при наличии ограничений

$$\boldsymbol{S_i \in \omega}, \qquad \boldsymbol{i = \overline{1, N}}.$$

В данной системе функция $\boldsymbol{F_1(o_i, o_j)}$ определяет площадь пересечения областей видимости роботов $\boldsymbol{o_i}$ и $\boldsymbol{o_j}$, функция $\boldsymbol{F_2(o_i, b_j)}$ определяет площадь пересечения области видимости робота $\boldsymbol{o_i}$ и препятствия $\boldsymbol{b_j}$, $\boldsymbol{\omega}$ задает множество допустимых положений робота на плоскости $\boldsymbol{S}$.

В задачу также входит создание программной компьютерной модели, способной имитировать двухмерное пространство с расположенными на нем препятствиями, которое являлось бы областью действия группы роботов. Кроме того, компьютерная модель должна демонстрировать эффективность работы алгоритма.

Глава 3. Обзор литературы

Разработки в области роевого интеллекта начались в США в середине 80-ых годов прошлого века. Первая компьютерная модель, заложившая фундаментальные принципы для роевых алгоритмов, была написана Крейгом Рейнольдсом в 1987 году [6].

Эта модель описывает поведение отдельных особей стаи птиц во время полета. Рейнольдс не столько ставил перед собой целью решение какой-либо практической задачи, сколько хотел продемонстрировать степень эффективности и правдоподобности, которую можно достичь, используя лишь небольшой набор простых правил: птицы должны избегать столкновения с препятствиями и двигаться в том же направлении и с той же скоростью, что и их соседи. Этого оказалось достаточно для построения реалистичной компьютерной модели, которая имитирует поведения всей стаи.

Результаты этой работы вызвали интерес со стороны многих исследователей. В 1995 году Рассел Эберхарт и Джеймс Кеннеди использовали модель Рейнольдсона для решения задачи оптимизации групповых усилий, которые прикладывают птицы во время полета, чтобы сохранять необходимую дистанцию между отдельными особями внутри стаи [7].

Эрик Бонабэу, Марко Дориго и Гай Тераулэз продолжили наблюдение за естественными биологическими системами с целью их адаптации для управления уже искусственными системами, такими как группы роботов [2].

В России задачами самоорганизующихся децентрализованных систем занимается Таганрогский НИИ Многопроцессорных Вычислительных Систем Имени Академика А. В. Каляева. Большую вклад в групповую робототехнику внесли И. А. Каляев, А. Р. Гайдук, Д. Я. Иванов и С. Г. Капустян [1, 5, 8, 9]. Ими были представлены теоретические обоснования для использования роевого алгоритма в конкретных практических задачах.

В частности, И. А. Каляевым и С. Г. Капустяном был создан и реализован алгоритм для решения задачи покрытия максимальной площади [9]. Однако в разработанном ими алгоритме препятствия играют относительно незначительную роль: при принятии решения роботы не учитывают ограничения, которые препятствия могут накладывать на область видимости.

Глава 4. Методы

4.1. Жадный алгоритм

Для решения задачи максимизации сплошного покрытия были созданы и программно реализованы два алгоритма, представляющие собой различные подходы к обозначенной проблеме.

Первый подход опирается на принципы, применяющиеся в жадных алгоритмах, которые являются распространенным методом решения задач оптимизации [10].

На каждом этапе работы данный алгоритм предпринимает решение, соответствующее локальному минимуму целевого функционала (2.1). Для работы жадному алгоритму необходимы сведения о месторасположении всех препятствий в обозначенной области и координаты всех роботов группы. Таким образом, жадный алгоритм может работать лишь в детерминированной среде.

В начале работы алгоритма на плоскости $\boldsymbol{S}$ задается начальная точка, вокруг которой роботы группы $\boldsymbol{O} = \{\boldsymbol{o_1} \dots \boldsymbol{o_N}\}$ будут выстраивать сплошное покрытие. Далее определяется ближайшая к начальной точке еще не наблюдаемая группой точка $\boldsymbol{S_0} \in \boldsymbol{S}$ и ближайший к этой точке робот $\boldsymbol{o_i}$. Согласно принципам жадного алгоритма, робот $\boldsymbol{o_i}$ выбирает новое местоположение $\boldsymbol{S_i}$ таким образом, чтобы точка $\boldsymbol{S_0}$ попадала в область его видимости, а пересечение области видимости робота $\boldsymbol{o_i}$ с препятствиями $\boldsymbol{b_1} \dots \boldsymbol{b_P}$ и областями видимости роботов, уже принявших решение о новом местоположении, было минимальным, то есть соответствовало локальному экстремуму функционала (2.1).

Роботы принимают решение последовательно. Работа алгоритма будет продолжаться, пока все роботы группы $\boldsymbol{O}$ не примут решение о новом местоположении.

Так как использование жадных алгоритмов является устоявшимся подходом к решению подобных задач, то результаты работы данного алгоритма могут служить своеобразным ориентиром для оценки результатов, полученных при помощи роевого интеллекта. Для обоих алгоритмов эта оценка будет измеряться в процентах и вычисляться по следующей формуле:

$$\mathbf{100} * \frac{\sum_{i=1}^{N}\left(\boldsymbol{\pi} * \boldsymbol{R^2} - \left(\sum_{j=i+1}^{N} \boldsymbol{F_1}(\boldsymbol{o_i}, \boldsymbol{o_j}) + \sum_{j=1}^{P} \boldsymbol{F_2}(\boldsymbol{o_i}, \boldsymbol{b_j})\right)\right)}{\boldsymbol{N} * \boldsymbol{\pi} * \boldsymbol{R^2}}. \quad (4.1)$$

Необходимо заметить, что такая оценка не может достигать значения в 100% при числе роботов $\boldsymbol{N > 2}$, так как условие пересечения областей видимости роботов необходимо для обеспечения сплошного покрытия, а значит числитель дроби будет всегда меньше знаменателя. Несмотря на это, данная оценка эффективности работы алгоритма является наглядной и удобной для использования.

4.2. Роевой интеллект

Роевой интеллект, способный решать задачу максимизации сплошного покрытия, был создан на основе итерационного алгоритма. На каждой итерации робот пытается уменьшить значение целевого функционала (2.1) путем отдаления от начальной точки и соседних роботов. Если же робот не образует с группой сплошного покрытия, то он начинает сближение с начальной точкой до тех пор, пока условие сплошного покрытия не будет выполнено.

Работа алгоритма будет продолжаться, пока изменение наблюдаемой области носит значительный характер, то есть превосходит какую-либо величину $\boldsymbol{\varepsilon}$, или не будет выполнено условие сплошного покрытия.

В данной реализации роевого интеллекта в качестве критерия остановки используется относительная погрешность решения. Роботы будут продолжать искать лучшее решение до тех пор, пока величина $\boldsymbol{100 * |S_2 - S_1|/S_2}$ больше наперед заданного процента. Тут $\boldsymbol{S_2}$ – наблюдаемая группой площадь на данной итерации работы алгоритма, $\boldsymbol{S_1}$ – площадь, наблюдаемая на предыдущей итерации алгоритма. Каляевым И. А. было установлено, что сходимость такого процесса носит линейный характер [5].

В отличие от жадного алгоритма, итерационный является асинхронным, то есть роботы действуют параллельно и независимо друг от друга. Взаимодействия между роботами группы $\boldsymbol{O}$ осуществляются на основе простых локальных правил и технического зрения. Помимо этого, алгоритм способен функционировать в недетерминированной среде, роботам не нужно знать расположение препятствий или других роботов заранее.

Таким образом, созданный алгоритм отвечает ключевым критериям роевого интеллекта: масштабируемость, асинхронность и простота локальных правил – а возможность функционирования в недетерминированной среде расширяет ряд решаемых задач.

Глава 5. Компьютерная реализация

5.1. Интерфейс программы

С помощью программной среды Microsoft Visual Studio была реализованная компьютерная модель поведения группы роботов в среде с препятствиями, способная демонстрировать эффективность работы различных алгоритмов. Модель представляет собой компьютерную программу с графическим интерфейсом, который представлен на рисунках 5.1 – 5.4.

При помощи графического интерфейса пользователь имеет возможность расположить на плоскости препятствия, задать численность группы роботов и радиус видимости каждого из них, затем выбрать начальную точку, вокруг которой роботы будут выстраиваться с целью создать максимальное сплошное покрытие. После чего пользователем выбирается одна из двух предложенных стратегий управления группой. Впоследствии пользователь может вернуть роботов в начальное положение или полностью сбросить решение и начальные данные.

Компьютерная модель отображает расположение препятствий на плоскости, а также расположение роботов и их областей видимости в процессе выполнения алгоритмов. В конце выполнения алгоритма программой отображаются результаты: наблюдаемая группой площадь, эффективность, количество итераций и время выполнения алгоритма.

Рис. 5.1: Стартовое окно программы

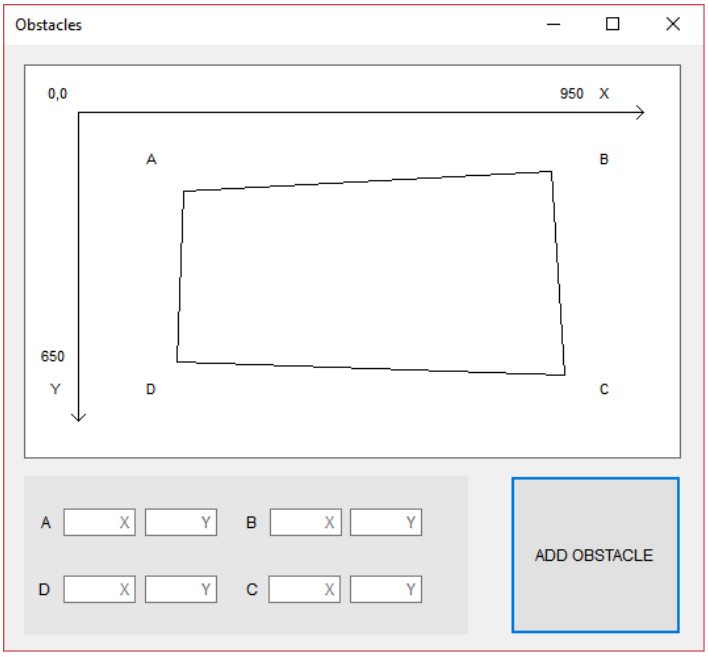

Рис. 5.2: Окно добавления препятствия

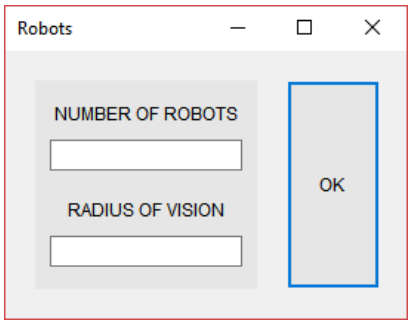

Рис. 5.3: Окно добавления группы роботов

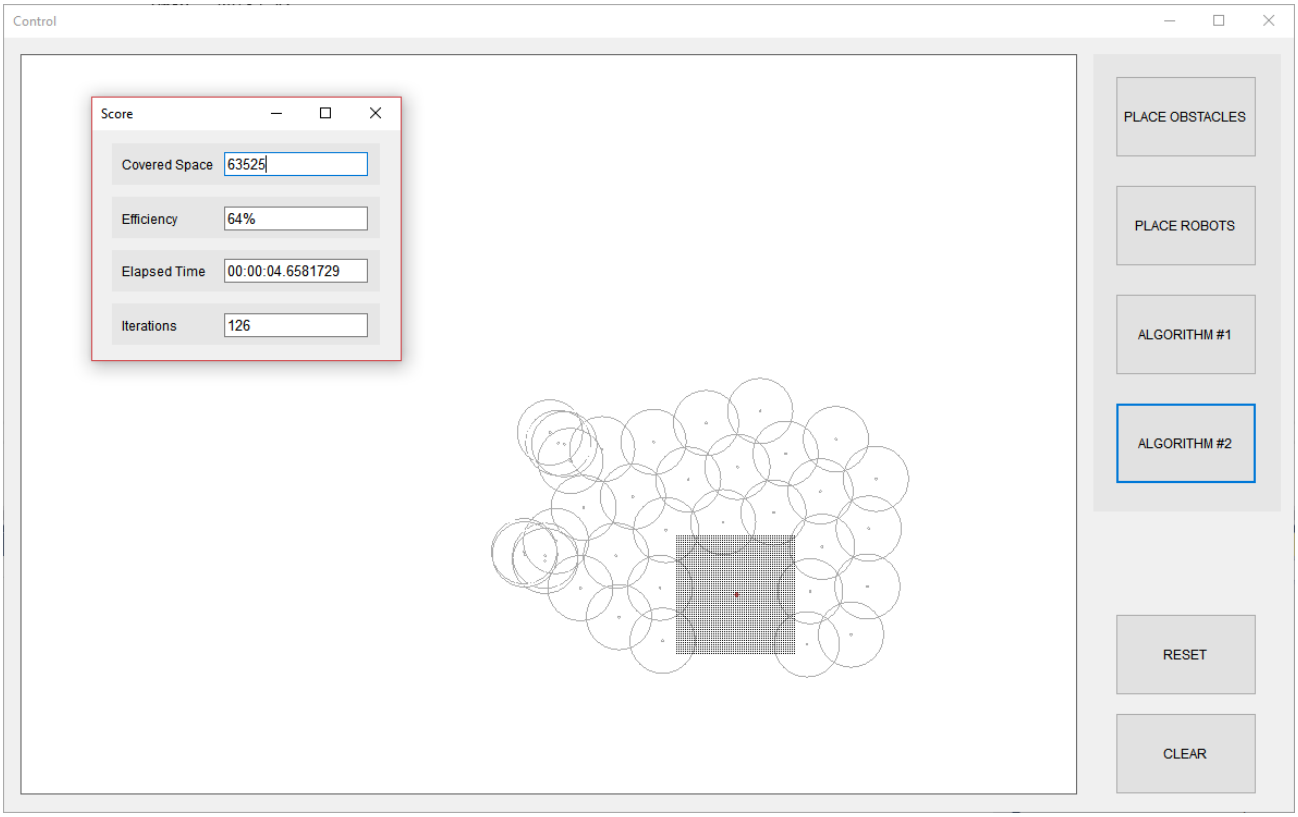

Рис. 5.4: Пример результата работы роевого алгоритма в среде с препятствием для группы численностью 35 роботов, начальная точка находится внутри препятствия

Как видно из рисунка 5.4, области видимости отдельных роботов отображаются окружностями, центром которых является сам робот, препятствия представлены закрашенными многоугольниками.

5.2. Структура программы

Алгоритм работы компьютерной модели записан в четырех файлах: Form1, Form2, Form3 и Form4 – исходный код которых приведен с комментариями в приложениях.

Файл Form1 содержит функции Algorithm1Button_Click и Algorithm2Button_Click, в которых записана программная реализация жадного и

роевого алгоритмов. Исходный код этих функций показан в приложениях 1 и 2 соответственно.

Помимо основных алгоритмов, файл Form1 содержит многие вспомогательные структуры и функции, использующиеся моделью для промежуточных вычислений. ObstaclesButton_Click вызывает окно добавления препятствий, PlaceButton_Click отвечает за отображение окна добавления группы роботов, PictureBox_Click позволяет добавить начальную точку, ResetButton_Click возвращает систему в начальное состояние, ClearButton_Click полностью обнуляет решение и соответствующие ему начальные данные. С исходным кодом этих функций можно ознакомится в приложении 3.

Функция AddObstaclesButton_Click в файле Form2 непосредственно добавляет препятствие в начальные условия компьютерной модели. Файл Form3 содержит функции SetScore, SetEfficiency, SetTime и SetIterations, с помощью которых осуществляется отображение результатов работы алгоритма. Функция AddButton_Click в файле Form4 добавляет группу роботов определенной численности и радиуса видимости в условия задачи. Исходный код для файлов Form2, Form3 и Form4 отображен в приложенияxs 4-6 соответственно.

5.3. Алгоритм работы с программой

Работа с программой осуществляется при помощи графического интерфейса и его интерактивных элементов, таких как кнопки, текстовые поля и графические области. Более подробно инструкция взаимодействия с программой описана на рисунке 5.5 при помощи блок-схемы.

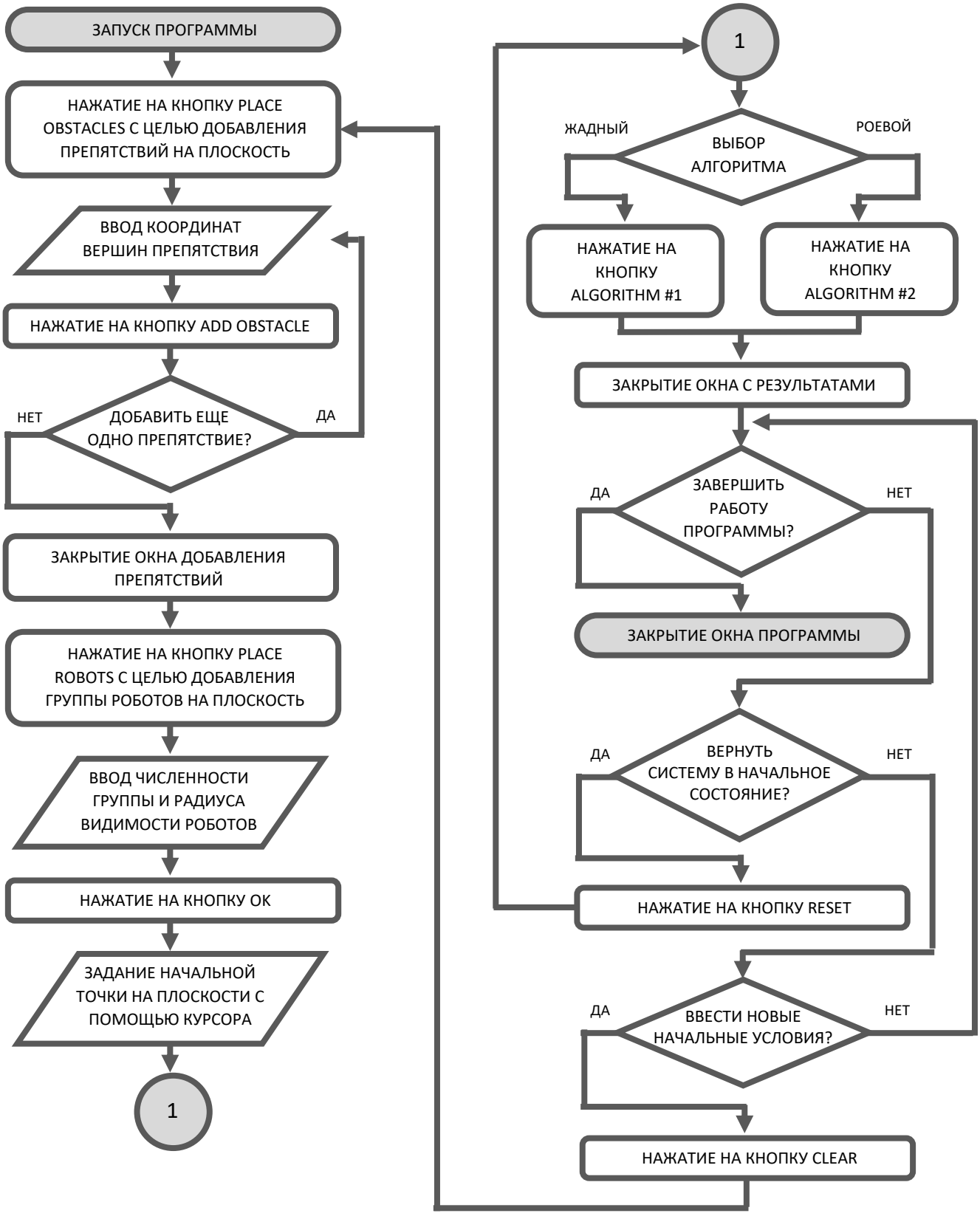

Рис. 5.5: Алгоритм работы с программой

Глава 6. Выводы

6.1. Результаты работы алгоритмов в моделях статичной среды

Работа обоих алгоритмов в статичной среде с неподвижными препятствиями и постоянным числом роботов проверялась при помощи созданной компьютерной модели. При проверке рассматривались несколько различных ситуаций, в которых варьировалось число роботов в группе, а также наличие или отсутствие препятствий на плоскости. Для роевого интеллекта дополнительно изменялся критерий остановки работы алгоритма.

Как было положено в главе 4, эффективность алгоритмов оценивалась по формуле (4.1). Каждая ситуация рассматривалась в серии из 10 опытов. В таблицах 6.1 – 6.4 зафиксированы значения, соответствующие минимуму и максимуму полученных результатов, также вычислены средние значения для всей серии опытов.

В статичной среде без препятствий начальные данные модели не влияют на результаты работы жадного алгоритма. Расположение роботов относительно начальной точки может лишь изменить последовательность принятия решений роботами, но принимаемые решения останутся теми же. Поэтому при моделировании серии опытов для одинакового числа роботов были получены идентичные результаты.

		Время	Эффективность
Кол-во роботов	30	1.3514 с.	68.42%
	60	3.1631 с.	63.21%
	120	8.5527 с.	62.36%

Табл. 6.1: Результаты работы жадного алгоритма в статичной среде без препятствий

В статичной среде с препятствиями начальные данные задачи уже влияют на результаты работы жадного алгоритма. Такие факторы, как расположение начальной точки относительно препятствий, их количество и координаты вершин, изменяют принимаемые роботами решения, поэтому для каждого эксперимента были получены уникальные данные.

		Время			Эффективность		
		Наименьшее	Наибольшее	Среднее	Наименьшая	Наибольшая	Средняя
Кол-во роботов	30	1.7151 с.	1.9309 с.	1.7517 с.	56.21%	62.34%	58.82%
	60	4.0801 с.	4.2011 с.	4.1276 с.	55.32%	63.35%	59.22%
	120	10.2756 с.	16.7549 с.	12.3628 с.	52.14%	61.12%	56.37%

Табл. 6.2: Результаты работы жадного алгоритма в статичной среде с препятствиями

		Время			Итерации			Эффективность		
		Наименьшее	Наибольшее	Среднее	Наименьшее	Наибольшее	Среднее	Наименьшая	Наибольшая	Средняя
Кол-во роботов	30	0.9498. с.	0.9821 с.	0.9675 с.	69 шт.	74 шт.	72 шт.	59.76%	72.31%	71.24%
	60	3.0137 с.	4.1762 с.	3.7554 с.	64 шт.	83 шт.	75 шт.	66.36%	70.12%	68.37%
	120	8.9377 с.	10.4793 с.	9.7586 с.	52 шт.	69 шт.	61 шт.	59.93%	67.51%	65.69%

Табл. 6.3: Результаты работы роевого алгоритма в статичной среде без препятствий. Критерий остановки: 1%

На результаты работы роевого алгоритма влияет начальное положение роботов. Так как при моделировании использовалось случайное распределение роботов, то результаты работы алгоритма для одинакового числа роботов разнятся даже в среде без препятствий.

		Время			Итерации			Эффективность		
		Наименьшее	Наибольшее	Среднее	Наименьшее	Наибольшее	Среднее	Наименьшая	Наибольшая	Средняя
Кол-во роботов	30	1.8502 с.	2.8013 с.	2.1021 с.	82 шт.	95 шт.	89 шт.	59.57%	68.57%	63.27%
	60	4.0568 с.	5.0519 с.	4.5861 с.	75 шт.	89 шт.	81 шт.	59.08%	64.51%	61.98%
	120	9.8977 с.	11.9845 с.	10.9822 с.	51 шт.	59 шт.	55 шт.	58.56%	64.85%	60.01%

Табл. 6.4: Результаты работы роевого алгоритма в статичной среде с препятствиями. Критерий остановки: 1%

Как видно из представленных таблиц, оба алгоритма обладают линейной сложностью, увеличение времени работы алгоритма пропорционально увеличению численности роботов. Появление препятствий в начальных условиях приводит к уменьшению оценки эффективности алгоритмов, что является закономерным следствием ограничений, которые препятствия могут накладывать на области видимости роботов.

Во всех рассмотренных ситуациях роевой алгоритм показал лучшую эффективность при сопоставимом или даже меньшем времени работы. Если положить, что жадный алгоритм завершает свою работу за единственную итерацию, то можно сделать вывод о меньшей требовательности роевого алгоритма к вычислительным устройствам робота, так как на выполнение одной итерации ему требуется значительно меньше времени.

6.2. Исследование критерия остановки роевого алгоритма

Рассмотренные дальше ситуации исследуют влияние критерия остановки роевого алгоритма на конечные результаты.

Использование более точного критерия остановки для роевого интеллекта ожидаемо привело к значительному увеличению числа итераций и необходимого для работы алгоритма времени. При этом наблюдается лишь небольшое улучшение оценки эффективности алгоритма, а ее разброс в серии опытов уменьшился. Результаты опытов приведены в таблицах 6.5 – 6.8.

		Время			Итерации			Эффективность		
		Наименьшее	Наибольшее	Среднее	Наименьшее	Наибольшее	Среднее	Наименьшая	Наибольшая	Средняя
Кол-во роботов	30	1.1412 с.	1.8242 с.	1.6134 с.	78 шт.	92 шт.	81 шт.	64.26%	75.11%	72.12%
	60	4.1267 с.	8.1499 с.	6.7957 с.	77 шт.	112 шт.	95 шт.	68.75%	72.15%	69.99%
	120	8.5974 с.	18.4898 с.	14.8791 с.	53 шт.	89 шт.	77 шт.	64.84%	68.92%	65.97%

Табл. 6.5: Результаты работы роевого алгоритма в статичной среде без препятствий. Критерий остановки: 0.1%

		Время			Итерации			Эффективность		
		Наименьшее	Наибольшее	Среднее	Наименьшее	Наибольшее	Среднее	Наименьшая	Наибольшая	Средняя
Кол-во роботов	30	1.4592 с.	3.9043 с.	1.9412 с.	85 шт.	111 шт.	91 шт.	60.52%	69.44%	64.41%
	60	3.6259 с.	7.9522 с.	5.5761 с.	78 шт.	116 шт.	99 шт.	60.57%	65.22%	63.25%
	120	11.5529 с.	14.1787 с.	12.2369 с.	62 шт.	74 шт.	68 шт.	60.07%	64.89%	61.29%

Табл. 6.6: Результаты работы роевого алгоритма в статичной среде с препятствиями. Критерий остановки: 0.1%

		Время			Итерации			Эффективность		
		Наименьшее	Наибольшее	Среднее	Наименьшее	Наибольшее	Среднее	Наименьшая	Наибольшая	Средняя
Кол-во роботов	30	2.2817 с.	15.3937 с.	6.9257 с.	109 шт.	175 шт.	148 шт.	66.47%	76.12%	72.58%
	60	19.5211 с.	25.3937 с.	21.1368 с.	217 шт.	302 шт.	241 шт.	68.92%	75.28%	73.77%
	120	27.3719 с.	59.7597 с.	38.5578 с.	110 шт.	229 шт.	163 шт.	65.87%	70.87%	69.19%

Табл. 6.7: Результаты работы роевого алгоритма в статичной среде без препятствий. Критерий остановки: 0.01%

		Время			Итерации			Эффективность		
		Наименьшее	Наибольшее	Среднее	Наименьшее	Наибольшее	Среднее	Наименьшая	Наибольшая	Средняя
Кол-во роботов	30	4.7437 с.	11.3898 с.	7.6578 с.	106 шт.	216 шт.	154 шт.	60.74%	70.41%	63.65%
	60	8.6116 с.	19.7682 с.	15.8733 с.	107 шт.	179 шт.	146 шт.	61.86%	67.08%	63.99%
	120	19.1242 с.	47.3144 с.	28.7347 с.	115 шт.	182 шт.	143 шт.	63.24%	70.28%	65.42%

Табл. 6.8: Результаты работы роевого алгоритма в статичной среде с препятствиями. Критерий остановки: 0.01%

6.3. Поведение роевого интеллекта в моделях динамической среды

Роевой интеллект способен решать поставленную задачу в недетерминированной динамической среде. Эта особенность алгоритма позволила смоделировать несколько интересных для рассмотрения ситуаций.

Первая ситуация имитировала выход из строя части группы роботов. В какой-то момент времени переставали функционировать роботы, оказавшиеся в некоторой области вокруг начальной точки.

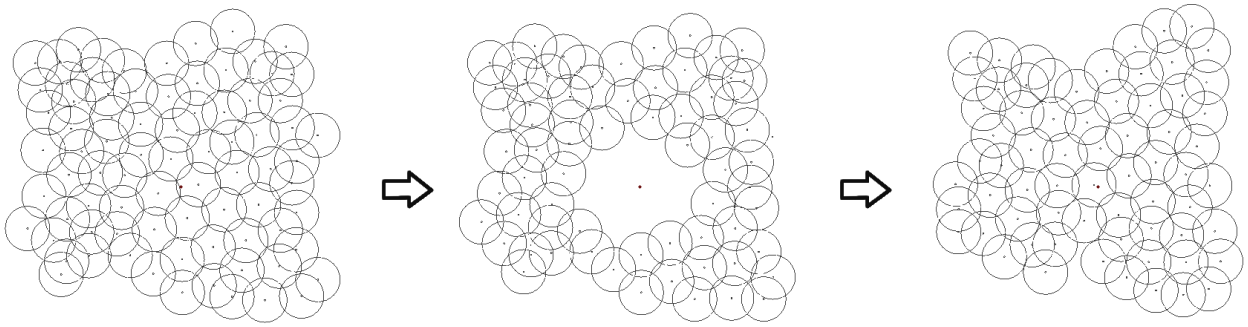

Рис. 6.1: Модель выхода из строя части роботов группы

Интерес представляла оставшаяся часть группы роботов. Как видно из рисунка 6.1, действия функционирующих роботов были направлены на заполнение пространства вокруг начальной точки, которое оказалось не наблюдаемым.

Вторая ситуация являлась моделью среды с динамическими препятствиями. В реальных условиях препятствия могут менять свое местоположение или изменяться в размерах и форме. Рисунок 6.2 показывает, как роботы смогли адаптироваться под новую конфигурацию препятствия.

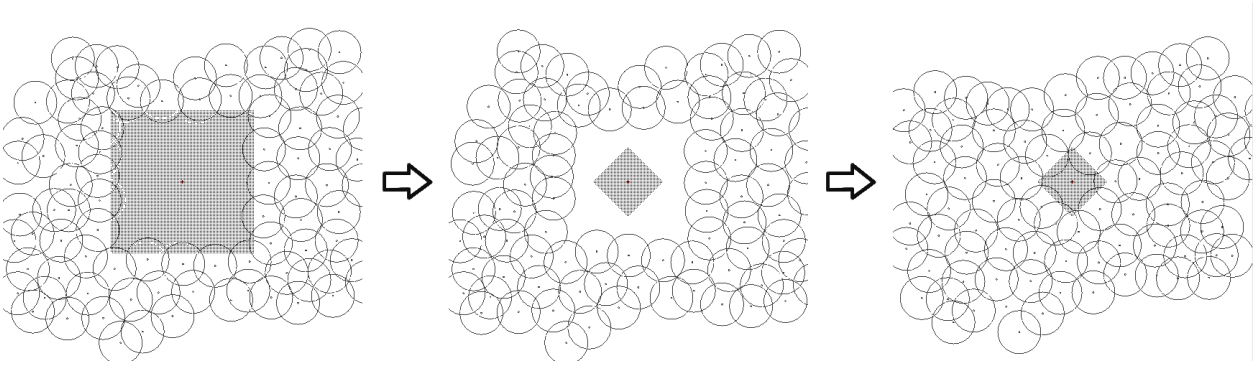

Рис. 6.2: Модель динамического препятствия

Еще одна рассмотренная ситуация позволила изучить поведение группы роботов при подвижной начальной точке. На рисунке 6.3 отображен процесс перемещения группы роботов вслед за движущейся начальной точкой. На протяжении всего время работы алгоритма группа сохраняла сплошное покрытие, а начальная точка оказывалась внутри наблюдаемой области.

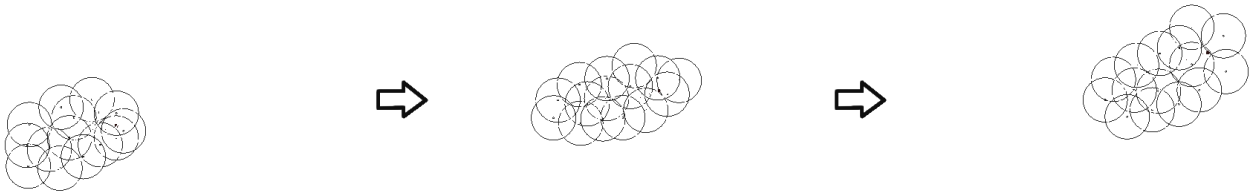

Рис.6.: Модель подвижной начальной точки

Поведение группы в представленных моделях показывает высокую степень адаптивности роевого интеллекта к постоянно меняющимся условиям динамической среды, что позволяет сделать вывод о возможности применения разработанного алгоритма на практике.

Глава 7. Заключение

При выполнении данной работы была изучена литература по соответствующей тематике и решены поставленные задачи, а именно:

- Разработан алгоритм, основанный на принципах роевого интеллекта и позволяющий решить задачу максимизации сплошного покрытия

- В программной среде Microsoft Visual Studio создана компьютерная модель, позволяющая имитировать работу алгоритма для группы роботов произвольной численности в среде с препятствиями, представленной двумерной плоскостью

- С помощью данной модели показана эффективность разработанного алгоритма

Полученные результаты демонстрируют перспективность использования роевого интеллекта для решения практических задач робототехники.

Глава 8. Список литературы

1. Иванов Д. Я. Использование принципов роевого интеллекта для управления целенаправленным поведением массово-применяемых микророботов в экстремальных условиях // Известия высших учебных заведений. Машиностроение, 2011. Вып. 9. С. 70-77.

2. Bonabeau E., Dorigo M., Theraulaz G. Swarm Intelligence. From Natural to Artificial Systems. New York, USA: Oxford University Press, 1999. 307 p.

3. Colorni A., Dorigo M., Maniezzo V. Distributed Optimization by Ant Colonies // European Conference on Artificial Life (ECAL91), 1991. P. 134-142.

4. Beni G. From Swarm Intelligence to Swarm Robotics // Lecture Notes in Computer Science, 2004. P. 1-9.

5. Каляев И. А., Гайдук А. Р., Капустян С. Г. Модели и Алгоритмы коллективного управления в группах роботов. М.: ФИЗМАТЛИТ, 2009. 280 с.

6. Reynolds C. Flocks, Herds, and Schools: A Distributed Behavioral Model // Computer Graphics (SIGGRAPH ’87), 1987. P. 25–34.

7. Kennedy J., Eberhart R. C. Particle Swarm Optimization // IEEE International Conference on Neural Networks, 1995. P. 1942-1948.

8. Каляев И. А., Гайдук А.Р. Стайные принципы управления в группе объектов // Искусственный интеллект, 2004. Т. 3. С. 700-708.

9. Капустян С. Г., Кулиничев Р. Н. Алгоритм и имитационная модель решения задачи оптимального покрытия поверхности группой роботов // Искусственный интеллект, 2004. Т. 2. С. 185-194.

10. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. Introduction to Algorithms, Third Edition. London, England: The MIT Press, 2009. 1312 p.

Приложения

Приложение 1

```
private void Algorithm1Button_Click(object sender, EventArgs e) //жадный
алгоритм
{
    Graphics g = this.pictureBox1.CreateGraphics();
    Pen PointPen = new Pen(Color.IndianRed, 2F);
    Brush myBrush = new SolidBrush(Color.CornflowerBlue);
    Brush VPBrush = new SolidBrush(Color.IndianRed);

    System.Diagnostics.Stopwatch myStopwatch = new
System.Diagnostics.Stopwatch(); //таймер для подсчета времени работы
алгоритма

    if (CRS.X < 0) //в случае, когда начальная точка не задана
пользователем, она задается как центр масс группы
    {
        double xmin = RB[0].X; double xmax = RB[0].X; double ymin =
RB[0].Y; double ymax = RB[0].Y;

        for (int i = 1; i < RB.Count; i++)
        {
            if (RB[i].X < xmin)
                xmin = RB[i].X;
            else if (RB[i].X > xmax)
                xmax = RB[i].X;

            if (RB[i].Y < ymin)
                ymin = RB[i].Y;
            else if (RB[i].Y > ymax)
                ymax = RB[i].Y;
        }

        CRS.X = (xmin + xmax) / 2;
        CRS.Y = (ymin + ymax) / 2;

        g.DrawEllipse(PointPen, (float)CRS.X - 1, (float)CRS.Y - 1, 2,
2);
    }

    List<Robo> VP = new List<Robo>(); //массив с вакантными точками,
которые необходимо наблюдать для сплошного покрытия
```

```
    Robo Fst = new Robo();

    for (int i = 0; i < OBS.Count; i++) //проверка, находится ли
начальная точка в препятствии
        if (Orient(OBS[i], CRS) == 0) //в таком случае за первую
вакантную точку принимается ближайшая не занятая препятствием
        {
            Robo temp = ClosestP(OBS[i], CRS, 0);
            Fst.X = temp.X + (temp.X - CRS.X) / Rast(temp, CRS);
            Fst.Y = temp.Y + (temp.Y - CRS.Y) / Rast(temp, CRS);
            VP.Add(Fst);
        }

    if (VP.Count == 0) //в противном случае происходит добавление
начальной точки в качестве первой вакантной в список VP
    { Fst = CRS; VP.Add(Fst); }

    for (int i = 0; i < RB.Count; i++)
    {
        System.Threading.Thread.Sleep(10);

        myStopwatch.Start();

        RSwap(RB, VP[0], i); //нахождение ближайшего к первой точке VP
робота из RB

        DrawRB(RB[i], OBS, R, this.pictureBox1, 0);

        double Smin = Math.PI * RB.Count * R * R; //площадь пересечения
видимости с другими роботами
        Robo Target = VP[0]; //точка, для которой эта площадь будет
минимальна в конечном итоге

        List<Robo> CloseVP = new List<Robo>(); //список с VP, которые
расположены на растоянии R друг от друга
        List<int> indexes = new List<int>(); //их индексы

        if (i != 2)
        {
            for (int j = 0; j < VP.Count; j++) //если вакантные точки
находятся рядом, имеет смысл закрыть их одним роботом
                if (Rast(VP[0], VP[j]) < R)
                { CloseVP.Add(VP[j]); indexes.Add(j); }
        }
        else
        { CloseVP.Add(VP[0]); indexes.Add(0); }
```

```
        bool possible = false; //проверка, можно ли наблюдать все точки
из списка CloseVP одновременно
        while (!possible)
        {
            for (double Fi = 0; Fi < 2 * Math.PI; Fi += Math.PI / 180)
                for (double Rad = 0; Rad < R; Rad++)
                {
                    PRobo PC = new PRobo(Fi, Rad, CloseVP[0]); //полярные
координаты точки относительно VP[0]
                    Robo DC = PtoD(PC); //декартовые координаты этой же
точки

                    bool check = false; //проверка на попадание в
препятствие
                    for (int z = 0; z < OBS.Count; z++)
                        if (Orient(OBS[z], DC) == 0)
                        {
                            check = true; break;
                        }

                    if (check)
                        break;

                    check = true; //проверка, находятся ли все точки
списка CloseVP в области видимости
                    for (int k = 0; k < OBS.Count; k++)
                        for (int j = 0; j < CloseVP.Count; j++)
                            if (ObCross(DC, CloseVP[j], OBS[k]))
                            { check = false; break; }

                    if (check)
                    {
                        possible = true;
                        double Scur = 0; //вычисление суммы площадей
пересечения областей видимости

                        for (int j = 0; j < i; j++)
                            Scur += PL(DC, RB[j], R); //пересечения с
областями видимости роботов

                        for (int z = 0; z < OBS.Count; z++) //пересечения
с препятствиями
                            if (Rast(DC, ClosestP(OBS[z], DC, 0)) < R)
                                Scur += IntSpace(OBS[z], DC, R);
```

```
                        if (Scur < Smin) //если полученная площадь
пересечения меньше минимальной на текущий момент, то присваиваем новое
значение целевой точке
                        {
                            Smin = Scur;
                            Target = DC;
                        }
                    }
                }
            if (!possible) //если не удается покрыть все вакантные точки
одним роботом, то удаляется наиболее отдаленная точка из списка CloseVP
            {
                double Rmax = 0; int jmax = 1;
                for (int j = 1; j < CloseVP.Count; j++)
                    if (Rast(CloseVP[0], CloseVP[j]) > Rmax)
                    { Rmax = Rast(CloseVP[0], CloseVP[j]); jmax = j; }
                CloseVP.RemoveAt(jmax); indexes.RemoveAt(jmax);
            }
        }

        Sfin += Math.PI * R * R - Smin;
        RB[i] = Target; //перемещение робота

        for (int z = CloseVP.Count - 1; z > -1; z--) //удаление точек
CloseVP из списка VP
        { g.FillRectangle(myBrush, (float)CloseVP[z].X - 2,
(float)CloseVP[z].Y - 2, 4, 4); VP.RemoveAt(indexes[z]); }

        CloseVP.Clear(); indexes.Clear();

        for (int j = 0; j < VP.Count; j++) //удаление других покрытых
точек роботом
            if (Rast(RB[i], VP[j]) < R + 2)
            {
                bool check = false;
                for (int z = 0; z < OBS.Count; z++)
                    if (ObCross(RB[i], VP[j], OBS[z]))
                    { check = true; break; }
                if (!check)
                { g.FillRectangle(myBrush, (float)VP[j].X - 2,
(float)VP[j].Y - 2, 4, 4); VP.RemoveAt(j); }
            }

        for (int j = 0; j < i; j++) //добавление в массив BT пересечений
с радиусами видимостей предыдущих роботов
            if (Rast(RB[i], RB[j]) < 2 * R)
            {
```

```
                double A = DtoP(RB[j], RB[i]).Fi;
                double dA = Math.Acos(Rast(RB[i], RB[j]) / (2 * R));
                PRobo PVP1 = new PRobo(A - dA, R, RB[i]); Robo VP1 =
PtoD(PVP1);
                PRobo PVP2 = new PRobo(A + dA, R, RB[i]); Robo VP2 =
PtoD(PVP2);

                bool fst = true; bool scnd = true;
                for (int k = 0; k < i; k++) //проверка, попадают ли точки
в уже наблюдаемую другими роботами область
                    if (k != j)
                    {
                        if (OBS.Count > 0)
                        {
                            if (Rast(VP1, RB[k]) < R + 2)
                                for (int z = 0; z < OBS.Count; z++)
                                    if (!ObCross(VP1, RB[k], OBS[z]))
                                        fst = false;
                            if (Rast(VP2, RB[k]) < R + 2)
                                for (int z = 0; z < OBS.Count; z++)
                                    if (!ObCross(VP2, RB[k], OBS[z]))
                                        scnd = false;
                        }
                        else
                        {
                            if (Rast(VP1, RB[k]) < R + 2)
                                fst = false;
                            if (Rast(VP2, RB[k]) < R + 2)
                                scnd = false;
                        }

                    }
                for (int k = 0; k < OBS.Count; k++) //проверка, попадают
ли точки в препятствия
                {
                    if (Rast(VP1, ClosestP(OBS[k], VP1, 0)) < 2)
                        fst = false;
                    if (Orient(OBS[k], VP1) == 0)
                        fst = false;
                    if (Rast(VP2, ClosestP(OBS[k], VP2, 0)) < 2)
                        scnd = false;
                    if (Orient(OBS[k], VP2) == 0)
                        scnd = false;
                }

                if (fst) VP.Add(VP1);
                if (scnd) VP.Add(VP2);
```

```
            }

        for (int j = 0; j < OBS.Count; j++) //добавление точек
пересечения с препятствиями
            if (Rast(RB[i], ClosestP(OBS[j], RB[i], 0)) < R)
            {
                bool fst = true; bool scnd = true;
                Robo VP1 = IntPoints(OBS[j], RB[i], R)[0];
                Robo VP2 = IntPoints(OBS[j], RB[i], R)[1];

                for (int k = 0; k < i; k++) //проверка, попадают ли точки
в уже наблюдаемую другими роботами область
                {
                    if (!ObCross(VP1, RB[k], OBS[j]))
                        if (Rast(VP1, RB[k]) < R + 3)
                            fst = false;
                    if (!ObCross(VP2, RB[k], OBS[j]))
                        if (Rast(VP2, RB[k]) < R + 3)
                            scnd = false;
                }

                for (int k = 0; k < OBS.Count; k++)  //проверка, попадают
ли точки в другие препятствия (если препятствия заданы некорректно)
                {
                    if (Orient(OBS[k], VP1) == 0)
                        fst = false;
                    if (Orient(OBS[k], VP2) == 0)
                        scnd = false;
                }

                if (fst) VP.Add(VP1);
                if (scnd) VP.Add(VP2);
            }

        if (VP.Count == 0) //после расположения самого первого робота
возможен случай, когда вакантная точка не определяется как пересечения с
другими препятствиями (тем более роботами)
        {
            Fst = RB[0];
            if (OBS.Count == 0)
                Fst.X += R;
            else
                for (double Fi = 0; Fi < 2 * Math.PI; Fi += Math.PI /
180)
                {
                    PRobo PC = new PRobo(Fi, R, Fst);
                    Robo DC = PtoD(PC);
```

```
                    for (int z = 0; z < OBS.Count; z++)
                        if (Orient(OBS[z], DC) != 0)
                        { Fst = DC; break; }
                }
            VP.Add(Fst);
        }

        if (VP.Count > 1) //переиндексация VP от самой ближней к
начальной точке до самой дальней
            for (int j = 0; j < VP.Count; j++)
                RSwap(VP, CRS, j);

        myStopwatch.Stop();

        DrawRB(RB[i], OBS, R, this.pictureBox1, 1);

        for (int j = 0; j < VP.Count; j++)
            g.FillRectangle(VPBrush, (float)VP[j].X - 2, (float)VP[j].Y -
2, 4, 4);

        System.Threading.Thread.Sleep(200);
    }

    Form3 form3 = new Form3();
    double Sfin2 = FinalScore(RB, OBS, R);
    form3.SetScore = Convert.ToString((int)Sfin2); //наблюдаемая площадь
    form3.SetEfficiency = Convert.ToString((int)(100 * Sfin2 / (RB.Count
* Math.PI * R * R))) + "%"; //использование потенциального ресурса группы
роботов
    form3.SetTime = Convert.ToString(myStopwatch.Elapsed); //время работы
алгоритма
    form3.SetIterations = "1"; //итерации
    form3.ShowDialog();
}
```

Приложение 2

```
private void Algorithm2Button_Click(object sender, EventArgs e)
//итерационный алгоритм
{
    Graphics g = this.pictureBox1.CreateGraphics();
    Pen myPen2 = new Pen(Color.CornflowerBlue, 1F);
    Pen myPen = new Pen(Color.Gray, 1F);
    Pen CRSPen = new Pen(Color.IndianRed, 2F);
    Pen EraserPen = new Pen(Color.White, 2F);
    Pen PointPen = new Pen(Color.IndianRed, 2F);

    System.Diagnostics.Stopwatch myStopwatch = new
System.Diagnostics.Stopwatch(); //таймер для подсчета времени работы
алгоритма

    if (CRS.X < 0) //в случае, когда начальная точка не задана
пользователем, она задается как центр масс группы
    {
        double xmin = RB[0].X; double xmax = RB[0].X; double ymin =
RB[0].Y; double ymax = RB[0].Y;

        for (int i = 1; i < RB.Count; i++)
        {
            if (RB[i].X < xmin)
                xmin = RB[i].X;
            else if (RB[i].X > xmax)
                xmax = RB[i].X;

            if (RB[i].Y < ymin)
                ymin = RB[i].Y;
            else if (RB[i].Y > ymax)
                ymax = RB[i].Y;
        }

        CRS.X = (xmin + xmax) / 2;
        CRS.Y = (ymin + ymax) / 2;

        g.DrawEllipse(PointPen, (float)CRS.X - 1, (float)CRS.Y - 1, 2,
2);
    }

    double halfsegment = 3 * Math.PI / 7; //половина рассматриваемого
сегмента
    double movestp = R / 5; //шаг
    int anglestp = 5; //дискретизация угла
```

```
    double S1 = 0; double S2 = Math.PI * R * R * RB.Count; //площади на
предыдущей и данной итерации
    List<bool> concheck = new List<bool>(); //проверка сплошного покрытия
    for (int i = 0; i < RB.Count; i++)
        concheck.Add(false);
    bool maincheck = false; //фиксация факта сплошного покрытия
    int iterations = 0; //счетчик итераций

    while (Math.Abs(S1 - S2) / S2 > 0.01 || !maincheck)
    {
        iterations++;
        maincheck = true;
        S1 = S2;
        for (int i = 0; i < RB.Count; i++)
        {
            //System.Threading.Thread.Sleep(1);
            g.DrawEllipse(EraserPen, (float)(RB[i].X - R),
(float)(RB[i].Y - R), (float)(2 * R), (float)(2 * R));
            g.DrawEllipse(EraserPen, (float)RB[i].X - 1, (float)RB[i].Y -
1, 2, 2);

            myStopwatch.Start();

            PRobo PCCRS = DtoP(CRS, RB[i]); //полярные координаты
начальной точки относительно робота
            double Fi1 = PCCRS.Fi - halfsegment; double Fi2 = PCCRS.Fi +
halfsegment;

            if (NablSegmentOBS(RB[i], Fi1, Fi2, RB, OBS, R, i)) //если
сегмент полностью в зоне видимости, попытаемся расширить покрываемую
площадь, отодвинув робота от центра
            {
                concheck[i] = true;

                List<Robo> CloseRB = new List<Robo>();

                for (int z = 0; z < RB.Count; z++) //составляем список
очень близко расположенных роботов
                    if (z != i)
                        if (Rast(RB[z], RB[i]) < R)
                            CloseRB.Add(RB[z]);

                if (CloseRB.Count > 0) //если появилась группа близко
расположенных роботов, пытаемся это исправить
                {
                    Robo MedP = new Robo(0, 0);
                    for (int z = 0; z < CloseRB.Count; z++)
```

```
                    { MedP.X += CloseRB[z].X / CloseRB.Count; MedP.Y +=
CloseRB[z].Y / CloseRB.Count; } //находим центр масс этой группы, от
которой робот будет удаляться

                    PCCRS = DtoP(MedP, RB[i]);
                }

                else //если такой группы нет, робот отдаляется от
центральной точки
                    PCCRS = DtoP(CRS, RB[i]);

                PRobo PCP = new PRobo(PCCRS.Fi + Math.PI, movestp,
RB[i]);
                for (double j = 0; j < anglestp; j++)
                {
                    PCP.Fi += Math.Pow((-1.0), j) * j * Math.PI /
(anglestp - 1); //полярные координаты рассматриваемой точки
                    Robo DCP = PtoD(PCP); //декартовые координаты этой же
точки

                    PRobo PCCRS2 = DtoP(CRS, DCP); //полярные координаты
начальной точки относительно рассматриваемой точки
                    Fi1 = PCCRS2.Fi - halfsegment; Fi2 = PCCRS2.Fi +
halfsegment;

                    if (NablSegmentOBS(DCP, Fi1, Fi2, RB, OBS, R, i) &&
NotOBS(DCP, OBS))
                    { RB[i] = DCP; break; }
                }
            }
            else //если сегмент не полностью в зоне видимости, попытаемся
исправить это
            {
                PRobo PCP = new PRobo(PCCRS.Fi, movestp, RB[i]);

                if (!NotOBS(PtoD(PCP), OBS))
                {
                    double FiColl = PCP.Fi;

                    while (!NotOBS(PtoD(PCP), OBS))
                    {
                        if (PCCRS.Fi < Math.PI / 2)
                            PCP.Fi += Math.PI / (anglestp - 1);
                        else if (PCCRS.Fi < Math.PI)
                            PCP.Fi -= Math.PI / (anglestp - 1);
                        else if (PCCRS.Fi < 3 * Math.PI / 2)
                            PCP.Fi += Math.PI / (anglestp - 1);
```

```
                        else
                            PCP.Fi -= Math.PI / (anglestp - 1);
                    }
                    RB[i] = PtoD(new PRobo(FiColl + Math.PI, movestp / 2,
RB[i])); //робот делает полшага назад
                }

                RB[i] = PtoD(new PRobo(PCP.Fi, movestp, RB[i]));
            }
            if (!concheck[i]) maincheck = false;
            myStopwatch.Stop();
            DrawRB(RB[i], OBS, R, this.pictureBox1, 1);
            g.DrawEllipse(CRSPen, (float)CRS.X - 1, (float)CRS.Y - 1, 2,
2);
        }

        S2 = FinalScore(RB, OBS, R);
        for (int j = 0; j < OBS.Count; j++)
            DrawOBS(OBS[j], this.pictureBox1);
    }

    Form3 form3 = new Form3();
    double Sfin2 = FinalScore(RB, OBS, R);
    form3.SetScore = Convert.ToString((int)Sfin2); //наблюдаемая площадь
    form3.SetEfficiency = Convert.ToString((int)(100 * Sfin2 / (RB.Count
* Math.PI * R * R))) + "%"; //использование потенциального ресурса группы
роботов
    form3.SetTime = Convert.ToString(myStopwatch.Elapsed); //время работы
алгоритма
    form3.SetIterations = Convert.ToString(iterations); //итерации
    form3.ShowDialog();
```

Приложение 3

```
namespace GroupControll
{
    public struct Robo //структура, задающая местоположение точки в ДПСК
    {
        public double X { get; set; }
        public double Y { get; set; }

        public Robo(double X, double Y)
        {
            this.X = X;
            this.Y = Y;
        }
    }

    public struct Obst //структура, задающая расположение препятствия
    {
        //вершины препятствия
        public Robo UL { get; set; } //верхняя левая
        public Robo UR { get; set; } //верхняя правая
        public Robo DR { get; set; } //нижняя правая
        public Robo DL { get; set; } //нижняя левая
        //уравнения прямых, задающих стороны
        //для горизонтальных y=kx+c
        public double UK { get; set; } //верхняя сторона
        public double UC { get; set; }
        public double DK { get; set; } //нижняя сторона
        public double DC { get; set; }
        //для вертикальных x=ky+c
        public double LK { get; set; } //левая сторона
        public double LC { get; set; }
        public double RK { get; set; } //правая сторона
        public double RC { get; set; }

        public Obst(Robo UL, Robo UR, Robo DR, Robo DL)
        {
            this.UL = UL;
            this.UR = UR;
            this.DR = DR;
            this.DL = DL;

            this.UK = (UR.Y - UL.Y) / (UR.X - UL.X);
            this.UC = UL.Y - UK * UL.X;
            this.DK = (DR.Y - DL.Y) / (DR.X - DL.X);
            this.DC = DL.Y - DK * DL.X;
```

```
            this.LK = (DL.X - UL.X) / (DL.Y - UL.Y);
            this.LC = UL.X - LK * UL.Y;
            this.RK = (DR.X - UR.X) / (DR.Y - UR.Y);
            this.RC = UR.X - RK * UR.Y;
        }
    }

    public struct PRobo //структура, задающая местоположение точки в
полярных координатах
    {
        public double Fi { get; set; } //угол, отсчитывающийся от
горизонтали по часовой стрелке
        public double Ra { get; set; } //радиус
        public Robo C { get; set; } //центр

        public PRobo(double Fi, double Ra, Robo C)
        {
            this.Fi = Fi;
            this.Ra = Ra;
            this.C = C;
        }
    }

    public partial class Form1 : Form
    {
        public static void Swap(List<Robo> RB, int A, int B)
//перестановка A и B элементов списка
        {
            Robo tmp = RB[A]; RB[A] = RB[B]; RB[B] = tmp;
        }

        public static void RSwap(List<Robo> RB, Robo P, int j)
//нахождение наближайшего к точке P робота из списка RB. Поиск
начинается с j позиции списка. В завершении функция переставляет
наближайшего робота на j-ую позицию
        {
            double Rmin = Rast(RB[j], P); int Nmin = j;

            for (int i = j+1; i < RB.Count; i++)
                if (Rast(RB[i], P) < Rmin)
                {
                    Rmin = Rast(RB[i], P);
                    Nmin = i;
                }

            Swap(RB, j, Nmin);
        }
```

```
        public static double Rast(Robo P1, Robo P2) //вычисление
расстояния между двумя точками
        {
            return Math.Sqrt((P1.X - P2.X) * (P1.X - P2.X) + (P1.Y -
P2.Y) * (P1.Y - P2.Y));
        }

        public static PRobo DtoP(Robo DC, Robo SC) //перевод декартовых
координат точки DC в полярные, за центр принимается точка SC
        {
            PRobo PC = new PRobo();

            PC.C = SC;
            PC.Ra = Rast(DC, SC); //радиус
            if (DC.Y - SC.Y > 0)
                PC.Fi = Math.Acos((DC.X - SC.X) / PC.Ra); //угол
отсчитывается по часовой от горизонтали
            else
                PC.Fi = 2 * Math.PI - Math.Acos((DC.X - SC.X) / PC.Ra);

            return PC;
        }

        public static Robo PtoD(PRobo PC) //перевод полярных координат в
декартовые
        {
            Robo DC = new Robo();

            DC.X = PC.C.X + PC.Ra * Math.Cos(PC.Fi);
            DC.Y = PC.C.Y + PC.Ra * Math.Sin(PC.Fi);

            return DC;
        }

        public static double PL(Robo P1, Robo P2, double R) //вычисление
площади пересечения областей видимости двух роботов с радиусом видимости
R
        {
            double F;

            if (Rast(P1, P2) < 2 * R)
            {
                F = 2 * Math.Acos(Rast(P1, P2) / (2 * R));

                return R * R * F - R * R * Math.Sin(F);
            }
```

```
            else
                return 0;
        }

        public static int Orient(Obst OB, Robo A) //определяет
местоположение робота A относительно препятствия OB
        {
            if (A.X * OB.UK + OB.UC < A.Y && A.X * OB.DK + OB.DC > A.Y &&
A.Y * OB.LK + OB.LC < A.X && A.Y * OB.RK + OB.RC > A.X)
                return 0; //внутри
            else if (A.X * OB.UK + OB.UC > A.Y && A.X * OB.DK + OB.DC >
A.Y)
            {
                if (A.Y * OB.LK + OB.LC > A.X && A.Y * OB.RK + OB.RC >
A.X)
                    return 1; //верх-лево
                else if (A.Y * OB.LK + OB.LC < A.X && A.Y * OB.RK + OB.RC
< A.X)
                    return 3; //верх-право
                else
                    return 2; //верх-середина
            }
            else if (A.X * OB.UK + OB.UC < A.Y && A.X * OB.DK + OB.DC <
A.Y)
            {
                if (A.Y * OB.LK + OB.LC > A.X && A.Y * OB.RK + OB.RC >
A.X)
                    return 7; //низ-лево
                else if (A.Y * OB.LK + OB.LC < A.X && A.Y * OB.RK + OB.RC
< A.X)
                    return 5; //низ-право
                else
                    return 6; //низ-середина
            }
            else
            {
                if (A.Y * OB.LK + OB.LC > A.X)
                    return 8; //середина-лево
                else
                    return 4; //середина-правоs
            }
        }

        public static Robo ClosestP(Obst OB, Robo A, int i) //нахождение
ближайшей к роботу A точки препятствия OB
        {
            if (i==1) //верхняя сторона
```

```
{
    Robo CL = new Robo();
    if (OB.UK == 0)
    {
        CL.X = A.X; CL.Y = CL.X * OB.UK + OB.UC;
    }
    else
    {
        double k1 = -1 / OB.UK;
        double c1 = A.Y - A.X * k1;

        CL.X = (c1 - OB.UC) / (OB.UK - k1);
        CL.Y = CL.X * k1 + c1;
    }

    if (CL.X > OB.UL.X && CL.X < OB.UR.X)
        return CL;
    else
    {
        if (Rast(CL, OB.UL) < Rast(CL, OB.UR))
            return OB.UL;
        else
            return OB.UR;
    }
}
else if (i==2) //правая сторона
{
    Robo CL = new Robo();
    if (OB.RK == 0)
    {
        CL.Y = A.Y; CL.X = A.X * OB.RK + OB.RC;
    }
    else
    {
        double k1 = -1 / OB.RK;
        double c1 = A.X - A.Y * k1;

        CL.Y = (c1 - OB.RC) / (OB.RK - k1);
        CL.X = CL.Y * k1 + c1;
    }

    if (CL.Y > OB.UR.Y && CL.Y < OB.DR.Y)
        return CL;
    else
    {
        if (Rast(CL, OB.UR) < Rast(CL, OB.DR))
            return OB.UR;
```

```
        else
            return OB.DR;
    }
}
else if (i == 3) //нижняя сторона
{
    Robo CL = new Robo();
    if (OB.DK == 0)
    {
        CL.X = A.X; CL.Y = CL.X * OB.DK + OB.DC;
    }
    else
    {
        double k1 = -1 / OB.DK;
        double c1 = A.Y - A.X * k1;

        CL.X = (c1 - OB.DC) / (OB.DK - k1);
        CL.Y = CL.X * k1 + c1;
    }

    if (CL.X > OB.DL.X && CL.X < OB.DR.X)
        return CL;
    else
    {
        if (Rast(CL, OB.DL) < Rast(CL, OB.DR))
            return OB.DL;
        else
            return OB.DR;
    }
}
else if (i == 4) //левая сторона
{
    Robo CL = new Robo();
    if (OB.LK == 0)
    {
        CL.Y = A.Y; CL.X = A.X * OB.LK + OB.LC;
    }
    else
    {
        double k1 = -1 / OB.LK;
        double c1 = A.X - A.Y * k1;

        CL.Y = (c1 - OB.LC) / (OB.LK - k1);
        CL.X = CL.Y * k1 + c1;
    }

    if (CL.Y > OB.UL.Y && CL.Y < OB.DL.Y)
```

```
                return CL;
            else
            {
                if (Rast(CL, OB.UL) < Rast(CL, OB.DL))
                    return OB.UL;
                else
                    return OB.DL;
            }
        }
        else //поиск ближайшей к роботу A точки по всему препятствию
        {
            Robo CL = OB.UL;
            for (int j=1; j<5; j++)
            {
                if (Rast(ClosestP(OB, A, j),A) < Rast(A, CL))
                    CL = ClosestP(OB, A, j);
            }
            return CL;
        }
    }

    public static List<Robo> IntPoints(Obst OB, Robo A, double R)
//нахождение точек пересечения области видимости радиуса R робота A с
препятствием OB
    {
        Robo P1 = new Robo(); Robo P2 = new Robo(); //точки
препятствия, ограничивающие видимость робота

        int i = Orient(OB, A);
        if (i == 0)
            { P1 = A; P2 = A; }
        else if (i == 1)
        {
            double b1 = OB.UK * OB.UC - A.X - A.Y * OB.UK;
            double c1 = A.X * A.X + A.Y * A.Y + OB.UC * OB.UC - R * R
- 2 * A.Y * OB.UC;
            double b2 = OB.LK * OB.LC - A.Y - A.X * OB.LK;
            double c2 = A.X * A.X + A.Y * A.Y + OB.LC * OB.LC - R * R
- 2 * A.X * OB.LC;
            P1.X = (-b1 + Math.Sqrt(b1 * b1 - c1 * (OB.UK * OB.UK +
1))) / (OB.UK * OB.UK + 1);
            P2.Y = (-b2 + Math.Sqrt(b2 * b2 - c2 * (OB.LK * OB.LK +
1))) / (OB.LK * OB.LK + 1);
            if (P1.X > OB.UR.X) P1.X = OB.UR.X; if (P2.Y > OB.DL.Y)
P2.Y = OB.DL.Y;
            P1.Y = P1.X * OB.UK + OB.UC; P2.X = P2.Y * OB.LK + OB.LC;
        }
```

```
            else if (i == 2)
            {
                double b = OB.UK * OB.UC - A.X - A.Y * OB.UK;
                double c = A.X * A.X + A.Y * A.Y + OB.UC * OB.UC - R * R
- 2 * A.Y * OB.UC;
                P1.X = (-b + Math.Sqrt(b * b - c * (OB.UK * OB.UK + 1)))
/ (OB.UK * OB.UK + 1);
                P2.X = (-b - Math.Sqrt(b * b - c * (OB.UK * OB.UK + 1)))
/ (OB.UK * OB.UK + 1);
                if (P1.X > OB.UR.X) P1.X = OB.UR.X; if (P2.X < OB.UL.X)
P2.X = OB.UL.X;
                P1.Y = P1.X * OB.UK + OB.UC; P2.Y = P2.X * OB.UK + OB.UC;
            }
            else if (i == 3)
            {
                double b1 = OB.RK * OB.RC - A.Y - A.X * OB.RK;
                double c1 = A.X * A.X + A.Y * A.Y + OB.RC * OB.RC - R * R
- 2 * A.X * OB.RC;
                double b2 = OB.UK * OB.UC - A.X - A.Y * OB.UK;
                double c2 = A.X * A.X + A.Y * A.Y + OB.UC * OB.UC - R * R
- 2 * A.Y * OB.UC;
                P1.Y = (-b1 + Math.Sqrt(b1 * b1 - c1 * (OB.RK * OB.RK +
1))) / (OB.RK * OB.RK + 1);
                P2.X = (-b2 - Math.Sqrt(b2 * b2 - c2 * (OB.UK * OB.UK +
1))) / (OB.UK * OB.UK + 1);
                if (P1.Y > OB.DR.Y) P1.Y = OB.DR.Y; if (P2.X < OB.UL.X)
P2.X = OB.UL.X;
                P1.X = P1.Y * OB.RK + OB.RC; P2.Y = P2.X * OB.UK + OB.UC;
            }
            else if (i == 4)
            {
                double b = OB.RK * OB.RC - A.Y - A.X * OB.RK;
                double c = A.X * A.X + A.Y * A.Y + OB.RC * OB.RC - R * R
- 2 * A.X * OB.RC;
                P1.Y = (-b + Math.Sqrt(b * b - c * (OB.RK * OB.RK + 1)))
/ (OB.RK * OB.RK + 1);
                P2.Y = (-b - Math.Sqrt(b * b - c * (OB.RK * OB.RK + 1)))
/ (OB.RK * OB.RK + 1);
                if (P1.Y > OB.DR.Y) P1.Y = OB.DR.Y; if (P2.Y < OB.UR.Y)
P2.Y = OB.UR.Y;
                P1.X = P1.Y * OB.RK + OB.RC; P2.X = P2.Y * OB.RK + OB.RC;
            }
            else if (i == 5)
            {
                double b1 = OB.DK * OB.DC - A.X - A.Y * OB.DK;
                double c1 = A.X * A.X + A.Y * A.Y + OB.DC * OB.DC - R * R
- 2 * A.Y * OB.DC;
```

```
                double b2 = OB.RK * OB.RC - A.Y - A.X * OB.RK;
                double c2 = A.X * A.X + A.Y * A.Y + OB.RC * OB.RC - R * R
- 2 * A.X * OB.RC;
                P1.X = (-b1 - Math.Sqrt(b1 * b1 - c1 * (OB.DK * OB.DK +
1))) / (OB.DK * OB.DK + 1);
                P2.Y = (-b2 - Math.Sqrt(b2 * b2 - c2 * (OB.RK * OB.RK +
1))) / (OB.RK * OB.RK + 1);
                if (P1.X < OB.DL.X) P1.X = OB.DL.X; if (P2.Y < OB.UR.Y)
P2.Y = OB.UR.Y;
                P1.Y = P1.X * OB.DK + OB.DC; P2.X = P2.Y * OB.RK + OB.RC;
            }
            else if (i == 6)
            {
                double b = OB.DK * OB.DC - A.X - A.Y * OB.DK;
                double c = A.X * A.X + A.Y * A.Y + OB.DC * OB.DC - R * R
- 2 * A.Y * OB.DC;
                P1.X = (-b - Math.Sqrt(b * b - c * (OB.DK * OB.DK + 1)))
/ (OB.DK * OB.DK + 1);
                P2.X = (-b + Math.Sqrt(b * b - c * (OB.DK * OB.DK + 1)))
/ (OB.DK * OB.DK + 1);
                if (P1.X < OB.DL.X) P1.X = OB.DL.X; if (P2.X > OB.DR.X)
P2.X = OB.DR.X;
                P1.Y = P1.X * OB.DK + OB.DC; P2.Y = P2.X * OB.DK + OB.DC;
            }
            else if (i == 7)
            {
                double b1 = OB.LK * OB.LC - A.Y - A.X * OB.LK;
                double c1 = A.X * A.X + A.Y * A.Y + OB.LC * OB.LC - R * R
- 2 * A.X * OB.LC;
                double b2 = OB.DK * OB.DC - A.X - A.Y * OB.DK;
                double c2 = A.X * A.X + A.Y * A.Y + OB.DC * OB.DC - R * R
- 2 * A.Y * OB.DC;
                P1.Y = (-b1 - Math.Sqrt(b1 * b1 - c1 * (OB.LK * OB.LK +
1))) / (OB.LK * OB.LK + 1);
                P2.X = (-b2 + Math.Sqrt(b2 * b2 - c2 * (OB.DK * OB.DK +
1))) / (OB.DK * OB.DK + 1);
                if (P1.Y < OB.UL.Y) P1.Y = OB.UL.Y; if (P2.X > OB.DR.X)
P2.X = OB.DR.X;
                P1.X = P1.Y * OB.LK + OB.LC; P2.Y = P2.X * OB.DK + OB.DC;
            }
            else
            {
                double b = OB.LK * OB.LC - A.Y - A.X * OB.LK;
                double c = A.X * A.X + A.Y * A.Y + OB.LC * OB.LC - R * R
- 2 * A.X * OB.LC;
                P1.Y = (-b + Math.Sqrt(b * b - c * (OB.LK * OB.LK + 1)))
/ (OB.LK * OB.LK + 1);
```

```
                P2.Y = (-b - Math.Sqrt(b * b - c * (OB.LK * OB.LK + 1)))
/ (OB.LK * OB.LK + 1);
                if (P1.Y > OB.DL.Y) P1.Y = OB.DL.Y; if (P2.Y < OB.UL.Y)
P2.Y = OB.UL.Y;
                P1.X = P1.Y * OB.LK + OB.LC; P2.X = P2.Y * OB.LK + OB.LC;
            }
            List<Robo> Points = new List<Robo>();
            Points.Add(P1); Points.Add(P2);
            return Points;
        }

        public static double IntSpace(Obst OB, Robo A, double R)
//вычисление площади пересечения области видимости радиуса R робота A с
препятствием OB
        {
            Robo P1 = IntPoints(OB, A, R)[0]; Robo P2 = IntPoints(OB, A,
R)[1];
            int i = Orient(OB, A);
            if (i == 0)
                return Math.PI * R * R;
            else if (i == 1)
            {
                double Fi;
                if (DtoP(P1, A).Fi < Math.PI)
                    Fi = DtoP(P2, A).Fi - DtoP(P1, A).Fi;
                else
                    Fi = DtoP(P2, A).Fi + 2 * Math.PI - DtoP(P1, A).Fi;
                double p1 = (Rast(P1, P2) + Rast(P1, A) + Rast(P2, A)) /
2;
                double p2 = (Rast(P1, P2) + Rast(P1, OB.UL) + Rast(P2,
OB.UL)) / 2;
                return R * R * Fi / 2 - Math.Sqrt(p1 * (p1 - Rast(P1,
P2)) * (p1 - Rast(P1, A)) * (p1 - Rast(P2, A))) + Math.Sqrt(p2 * (p2 -
Rast(P1, P2)) * (p2 - Rast(P1, OB.UL)) * (p2 - Rast(P2, OB.UL)));
            }
            else if (i == 2)
            {
                double Fi = DtoP(P2, A).Fi - DtoP(P1, A).Fi;
                double p = (Rast(P1, P2) + Rast(P1, A) + Rast(P2, A)) /
2;
                return R * R * Fi / 2 - Math.Sqrt(p * (p - Rast(P1, P2))
* (p - Rast(P1, A)) * (p - Rast(P2, A)));
            }
            else if (i == 3)
            {
                double Fi = DtoP(P2, A).Fi - DtoP(P1, A).Fi;
```

```
                double p1 = (Rast(P1, P2) + Rast(P1, A) + Rast(P2, A)) /
2;
                double p2 = (Rast(P1, P2) + Rast(P1, OB.UR) + Rast(P2,
OB.UR)) / 2;
                return R * R * Fi / 2 - Math.Sqrt(p1 * (p1 - Rast(P1,
P2)) * (p1 - Rast(P1, A)) * (p1 - Rast(P2, A))) + Math.Sqrt(p2 * (p2 -
Rast(P1, P2)) * (p2 - Rast(P1, OB.UR)) * (p2 - Rast(P2, OB.UR)));
            }
            else if (i == 4)
            {
                double Fi = DtoP(P2, A).Fi - DtoP(P1, A).Fi;
                double p = (Rast(P1, P2) + Rast(P1, A) + Rast(P2, A)) /
2;
                return R * R * Fi / 2 - Math.Sqrt(p * (p - Rast(P1, P2))
* (p - Rast(P1, A)) * (p - Rast(P2, A)));
            }
            else if (i == 5)
            {
                double Fi = DtoP(P2, A).Fi - DtoP(P1, A).Fi;
                double p1 = (Rast(P1, P2) + Rast(P1, A) + Rast(P2, A)) /
2;
                double p2 = (Rast(P1, P2) + Rast(P1, OB.DR) + Rast(P2,
OB.DR)) / 2;
                return R * R * Fi / 2 - Math.Sqrt(p1 * (p1 - Rast(P1,
P2)) * (p1 - Rast(P1, A)) * (p1 - Rast(P2, A))) + Math.Sqrt(p2 * (p2 -
Rast(P1, P2)) * (p2 - Rast(P1, OB.DR)) * (p2 - Rast(P2, OB.DR)));
            }
            else if (i == 6)
            {
                double Fi;
                if (DtoP(P2, A).Fi > Math.PI)
                    Fi = DtoP(P2, A).Fi - DtoP(P1, A).Fi;
                else
                    Fi = DtoP(P2, A).Fi + 2 * Math.PI - DtoP(P1, A).Fi;
                double p = (Rast(P1, P2) + Rast(P1, A) + Rast(P2, A)) /
2;
                return R * R * Fi / 2 - Math.Sqrt(p * (p - Rast(P1, P2))
* (p - Rast(P1, A)) * (p - Rast(P2, A)));
            }
            else if (i == 7)
            {
                double Fi;
                if (DtoP(P2, A).Fi < Math.PI)
                    Fi = DtoP(P2, A).Fi + 2 * Math.PI - DtoP(P1, A).Fi;
                else
                    Fi = DtoP(P2, A).Fi - DtoP(P1, A).Fi;
```

```
                double p1 = (Rast(P1, P2) + Rast(P1, A) + Rast(P2, A)) /
2;
                double p2 = (Rast(P1, P2) + Rast(P1, OB.DL) + Rast(P2,
OB.DL))/2;
                return R * R * Fi / 2 - Math.Sqrt(p1 * (p1 - Rast(P1,
P2)) * (p1 - Rast(P1, A)) * (p1 - Rast(P2, A))) + Math.Sqrt(p2 * (p2 -
Rast(P1, P2)) * (p2 - Rast(P1, OB.DL)) * (p2 - Rast(P2, OB.DL)));
            }
            else
            {
                double Fi;
                if (DtoP(P1, A).Fi < Math.PI)
                    Fi = DtoP(P1, A).Fi + 2 * Math.PI - DtoP(P2, A).Fi;
                else
                    Fi = DtoP(P1, A).Fi - DtoP(P2, A).Fi;
                double p = (Rast(P1, P2) + Rast(P1, A) + Rast(P2, A)) /
2;
                return R * R * Fi / 2 - Math.Sqrt(p * (p - Rast(P1, P2))
* (p - Rast(P1, A)) * (p - Rast(P2, A)));
            }
        }

        public static bool ObCross(Robo A, Robo B, Obst OB) //функция
вовзращает true, если между точками A и B находится препятствие OB
        {
            if (A.X == B.X) //случай вертикального отрезка
            {
                if (A.Y > B.Y)
                    { Robo temp = A; A = B; B = temp; }

                double YP;
                YP = A.X * OB.UK + OB.UC; //проверка пересечения с
верхней стороной
                if (A.Y - 2 < YP && YP < B.Y - 2)
                    if (A.Y + 2 < YP && YP < B.Y + 2)
                        return true;
                YP = A.X * OB.DK + OB.DC; //с нижней
                if (A.Y - 2 < YP && YP < B.Y - 2)
                    if (A.Y + 2 < YP && YP < B.Y + 2)
                        return true;
                if (OB.LK != 0)
                {
                    YP = (A.X - OB.LC) / OB.LK;
                    if (A.Y - 2 < YP && YP < B.Y - 2)
                        if (A.Y + 2 < YP && YP < B.Y + 2)
                            return true;
                }
```

```
                if (OB.RK != 0)
                {
                    YP = (A.X - OB.RC) / OB.RK;
                    if (A.Y - 2 < YP && YP < B.Y - 2)
                        if (A.Y + 2 < YP && YP < B.Y + 2)
                            return true;
                }
                return false;

            }
            else
            {
                if (A.X > B.X)
                    { Robo temp = A; A = B; B = temp; }
                double k = (A.Y - B.Y) / (A.X - B.X);
                double c = A.Y - A.X * k;

                double XP;
                if (k != OB.UK)
                {
                    XP = (OB.UC - c) / (k - OB.UK); //проверка
пересечения с верхней стороной
                    if (A.X - 2 < XP && XP < B.X - 2)
                        if (A.X + 2 < XP && XP < B.X + 2)
                            return true;
                }
                if (k != OB.DK)
                {
                    XP = (OB.DC - c) / (k - OB.DK); // с нижней
                    if (A.X - 2 < XP && XP < B.X - 2)
                        if (A.X + 2 < XP && XP < B.X + 2)
                            return true;
                }
                XP = ((k * OB.LC + c) / (1 - k * OB.LK) - c) / k; //c
левой
                if (A.X - 2 < XP && XP < B.X - 2)
                    if (A.X + 2 < XP && XP < B.X + 2)
                        return true;
                XP = ((k * OB.RC + c) / (1 - k * OB.RK) - c) / k; //c
правой
                if (A.X - 2 < XP && XP < B.X - 2)
                    if (A.X + 2 < XP && XP < B.X + 2)
                        return true;
                return false; //нет пересечений
            }
        }
```

```
        public static bool NablSegmentOBS(Robo A, double Fi1, double Fi2,
List<Robo> RB, List<Obst> OB, double R, int i) //функция возвращает true,
если сегмент области видимости робота A, заключенный между углами Fi1 и
Fi2, полностью находится в зоне видимости других роботов, номер i
позволяет не включать в рассмотрение i-ого робота списка RB
        {
            bool concheck = false;
            double step = 2;
            double anglestep = Math.PI / 90;
            for (double Fi = Fi1; Fi < Fi2; Fi += anglestep)
            {
                bool check = false; //проверка, находится ли точки в зоне
видимости другого робота или в препятствии
                for (double Rad = 0; Rad < R; Rad += step)
                {
                    PRobo PCP = new PRobo(Fi, Rad, A); //полярные
координаты проверяемой точки
                    if (!NotOBS(PtoD(PCP), OB))
                    { check = true; break; }
                }
                if (!check)
                {
                    PRobo PCP = new PRobo(Fi, R, A);
                    for (int j = 0; j < RB.Count; j++)
                        if (j != i && Rast(PtoD(PCP), RB[j]) < R)
                        { check = true; concheck = true; break; }
                }
                if (!check) return false; //точка не попадает в чужую
обл. видимости и сегмент находится в чужой области видимости неполностью
            }
            if (!concheck)
            {
                if (Fi2 < 2 * Math.PI) Fi1 += 2 * Math.PI;
                else Fi2 -= 2 * Math.PI;

                for (double Fi = Fi2; Fi < Fi1; Fi += 2 * anglestep)
                {
                    bool check = false; //проверка на препятствие
                    for (double Rad = 0; Rad < R; Rad += 4 * step)
                    {
                        PRobo PCP = new PRobo(Fi, Rad, A); //полярные
координаты проверяемой точки
                        if (!NotOBS(PtoD(PCP), OB))
                        { check = true; break; }
                    }
                    if (!check)
                    {
```

```
                        PRobo PCP = new PRobo(Fi, R, A);
                        for (int j = 0; j < RB.Count; j++)
                            if (j != i && Rast(PtoD(PCP), RB[j]) < R)
                            { concheck = true; break; } //робот
"соединен" с другим роботом
                    }
                    if (concheck) break; //сегмент находится в чужой
области видимости неполностью
                }
            }
                if (!concheck) return false;
            else return true;
        }

        public static double FinalScore(List<Robo> RB, List <Obst> OB,
double R) //возвращает наблюдаемую группой площадь
        {
            double LX, RX, UY, DY; long Score = 0;
            LX = RB[0].X; RX = RB[0].X; UY = RB[0].Y; DY = RB[0].Y;
            for (int i = 1; i < RB.Count; i++)
            {
                if (RB[i].X < LX) LX = RB[i].X;
                if (RB[i].X > RX) RX = RB[i].X;
                if (RB[i].Y < UY) UY = RB[i].Y;
                if (RB[i].Y > DY) DY = RB[i].Y;
            }

            LX -= R; RX += R; UY -= R; DY += R;
            double step = R/10;

            for (double X = LX; X < RX; X += step)
                for (double Y = UY; Y < DY; Y +=step)
                {
                    bool check2 = false; //true = попадание в препятствие
                    bool check = true; //true = ненаблюдаемая точка
                    Robo P = new Robo(X, Y);
                    for (int j = 0; j < OB.Count; j++)
                        if (Orient(OB[j],P) == 0)
                        { check2 = true; break; }
                    if (!check2)
                        for (int j = 0; j < RB.Count; j++)
                            if (Rast(P, RB[j]) < R)
                                { check = false;  break; }
                    if (check) Score++;
                }
            return (RX - LX) * (DY - UY) * (1 - Score / (((RX - LX) /
step) * ((DY - UY) / step)));
```

```
        }

        public static void DrawOBS(Obst OB,
System.Windows.Forms.PictureBox PB) //рисует препятствие
        {

            Graphics g = PB.CreateGraphics();
            Brush myBrush = new SolidBrush(Color.Black);

            float xmin, xmax, ymin, ymax;
            if (OB.UL.X < OB.DL.X) xmin = (float)OB.UL.X;
            else xmin = (float)OB.DL.X;
            if (OB.UR.X > OB.DR.X) xmax = (float)OB.UR.X;
            else xmax = (float)OB.DR.X;
            if (OB.UL.Y < OB.UR.Y) ymin = (float)OB.UL.Y;
            else ymin = (float)OB.UR.Y;
            if (OB.DL.Y > OB.DR.Y) ymax = (float)OB.DL.Y;
            else ymax = (float)OB.DR.Y;

            for (float i = xmin + 1; i < xmax; i += 2)
                for (float j = ymin + 1; j < ymax; j += 2)
                    if (j > i * OB.UK + OB.UC && j < i * OB.DK + OB.DC &&
i > j * OB.LK + OB.LC && i < j * OB.RK + OB.RC)
                        g.FillRectangle(myBrush, i, j, 1, 1);
        }

        public static void DrawRB(Robo A, List<Obst> OBS, double R,
System.Windows.Forms.PictureBox PB, int k) //рисует робота
        {
            Graphics g = PB.CreateGraphics();
            Pen Black = new Pen(Color.DarkGray, 1.5F);
            Pen WhiteP = new Pen(Color.White, 1.5F);
            Brush Blue = new SolidBrush(Color.CornflowerBlue);
            Brush WhiteB = new SolidBrush(Color.White);

            if (k == 0) //удаление прорисованных роботов
            {
                g.DrawEllipse(WhiteP, (float)(A.X - R), (float)(A.Y - R),
(float)(2 * R), (float)(2 * R)); //прорисовка центра робота и радиуса
видимости
                g.DrawEllipse(WhiteP, (float)(A.X - 1), (float)(A.Y - 1),
2, 2);
            }
            else //прорисовка роботов
            {
```

```
                g.DrawEllipse(Black, (float)(A.X - R), (float)(A.Y - R),
(float)(2 * R), (float)(2 * R)); //прорисовка центра робота и радиуса
видимости
                g.DrawEllipse(Black, (float)(A.X - 1), (float)(A.Y - 1),
2, 2);
            }
        }

        public static bool NotOBS(Robo A, List<Obst> OB) //возвращает
true, если точка не внутри какого-либо препятствия
        {
            for (int i = 0; i < OB.Count; i++)
                if (Orient(OB[i], A) == 0)
                    return false;
            return true;
        }

        int N = 0; //число роботов
        double R = 0; //радиус
        double Sfin = 0; //наблюдаемая площадь

        public List<Obst> OBS = new List<Obst>(); //список с
препятствиями
        List<Robo> RB = new List<Robo>(); //список с координатами роботов
        List<Robo> OldRB = new List<Robo>(); //начальное положение
роботов
        Robo CRS = new Robo(-1, -1); //положение курсора

        Random ran = new Random();

        public Form1()
        {
            InitializeComponent();
        }

        private void PlaceButton_Click(object sender, EventArgs e)
//расположение N роботов случайным образом по карте с учетом препятствий
        {
            Form4 form4 = new Form4();
            form4.Owner = this;
            form4.ShowDialog();
            if (form4.GetNumber != 0) N = form4.GetNumber;
            if (form4.GetRadius != 0) R = form4.GetRadius;
            if (form4.GetRobots.Count != 0) RB = form4.GetRobots;
            for (int i = 0; i < RB.Count; i++)
                OldRB.Add(RB[i]);
        }
```

```
        private void ResetButton_Click(object sender, EventArgs e)
//возврат роботов в начальное положение
        {
            Graphics g = this.pictureBox1.CreateGraphics();
            g.Clear(Color.White);
            for (int i = 0; i < OBS.Count; i++)
                DrawOBS(OBS[i], this.pictureBox1);
            for (int i = 0; i < RB.Count; i++)
                RB[i] = OldRB[i];
            for (int i = 0; i < RB.Count; i++)
                DrawRB(RB[i], OBS, R, this.pictureBox1, 1);
            Sfin = 0;
        }

        private void pictureBox1_Click(object sender, MouseEventArgs e)
//задание начальной точки с помощью курсора
        {
            Graphics g = this.pictureBox1.CreateGraphics();
            Pen Eraser = new Pen(Color.White, 2F);
            Pen myPen = new Pen(Color.IndianRed, 2F);
            g.DrawEllipse(Eraser, (float)CRS.X - 1, (float)CRS.Y - 1, 2,
2);

            CRS.X = e.X; CRS.Y = e.Y;

            g.DrawEllipse(myPen, (float)CRS.X - 1, (float)CRS.Y - 1, 2,
2);
        }

        private void ObstaclesButton_Click(object sender, EventArgs e)
//добавление препятствий
        {
            Form2 form2 = new Form2();
            form2.Owner = this;
            form2.ShowDialog();
        }

        private void ClearButton_Click(object sender, EventArgs e)
//сброс данных
        {
            Graphics g = this.pictureBox1.CreateGraphics();
            g.Clear(Color.White); RB.Clear(); OBS.Clear();
            CRS = new Robo(-1, -1); Sfin = 0;
        }
    }
}
```

Приложение 4

```
namespace GroupControll
{
    public partial class Form2 : Form
    {
        Random ran = new Random();
        double AX, AY, BX, BY, CX, CY, DX, DY; //координаты вершин
препятствия

        public Form2()
        {
            InitializeComponent();
            pictureBox1.Paint += pictureBox1_Paint;
        }

        private void pictureBox1_Paint(object sender, PaintEventArgs e)
        {
            Pen myPen = new Pen(Color.Black, 1.5F);
            e.Graphics.DrawLine(myPen, 40, 35, 470, 35);
            e.Graphics.DrawLine(myPen, 470, 35, 465, 30);
            e.Graphics.DrawLine(myPen, 470, 35, 465, 40);
            e.Graphics.DrawLine(myPen, 40, 35, 40, 270);
            e.Graphics.DrawLine(myPen, 40, 270, 35, 265);
            e.Graphics.DrawLine(myPen, 40, 270, 45, 265);
            e.Graphics.DrawLine(myPen, 120, 95, 400, 80);
            e.Graphics.DrawLine(myPen, 400, 80, 410, 235);
            e.Graphics.DrawLine(myPen, 410, 235, 115, 225);
            e.Graphics.DrawLine(myPen, 115, 225, 120, 95);
        } //прорисовка элементов интерфейса

        private void AXtextBox_TextChanged(object sender, EventArgs e)
//добавление вершин препятствия посредством текстовых полей
        { if (AXtextBox.Text.Length != 0) AX =
Convert.ToDouble(AXtextBox.Text); }

        private void AYtextBox_TextChanged(object sender, EventArgs e)
        { if (AYtextBox.Text.Length != 0) AY =
Convert.ToDouble(AYtextBox.Text); }

        private void BXtextBox_TextChanged(object sender, EventArgs e)
        { if (BXtextBox.Text.Length != 0) BX =
Convert.ToDouble(BXtextBox.Text); }

        private void BYtextBox_TextChanged(object sender, EventArgs e)
```

```
        { if (BYtextBox.Text.Length != 0) BY =
Convert.ToDouble(BYtextBox.Text); }

        private void CXtextBox_TextChanged(object sender, EventArgs e)
        { if (CXtextBox.Text.Length != 0) CX =
Convert.ToDouble(CXtextBox.Text); }

        private void CYtextBox_TextChanged(object sender, EventArgs e)
        { if (CYtextBox.Text.Length != 0) CY =
Convert.ToDouble(CYtextBox.Text); }

        private void DXtextBox_TextChanged(object sender, EventArgs e)
        { if (DXtextBox.Text.Length != 0) DX =
Convert.ToDouble(DXtextBox.Text); }

        private void DYtextBox_TextChanged(object sender, EventArgs e)
        { if (DYtextBox.Text.Length != 0) DY =
Convert.ToDouble(DYtextBox.Text); }

        private void AddObstaclesButton_Click(object sender, EventArgs e)
        {
            Form1 main = this.Owner as Form1;
            Robo A = new Robo(AX, AY);
            Robo B = new Robo(BX, BY);
            Robo C = new Robo(CX, CY);
            Robo D = new Robo(DX, DY);

            if (A.Y > C.Y) //перестановка вершин в случае их
неправильного занесения в форму
                { Robo temp = A; A = C; C = temp; }
            if (A.Y > D.Y)
                { Robo temp = A; A = D; D = temp; }
            if (B.Y > C.Y)
                { Robo temp = B; B = C; C = temp; }
            if (B.Y > D.Y)
                { Robo temp = B; B = D; D = temp; }
            if (A.X > B.X)
                { Robo temp = A; A = B; B = temp; }
            if (D.X > C.X)
                { Robo temp = D; D = C; C = temp; }

            Obst OB = new Obst(A, B, C, D);
            main.OBS.Add(OB); //добавление препятствия в список
            Form1.DrawOBS(OB, main.pictureBox1);
        }
    }
}
```

Приложение 5

```
namespace GroupControll
{

    public partial class Form3 : Form //отображение результатов работы
алгоритма
    {
        public string SetScore
        {   set { ScoretextBox.Text = value; }  }

        public string SetEfficiency
        {   set { EfficiencytextBox.Text = value; } }

        public string SetTime
        { set { TimetextBox.Text = value; } }

        public string SetIterations
        { set { IterationtextBox.Text = value; } }

        public Form3()
        {
            InitializeComponent();
        }
     }
}
```

Приложение 6

```
namespace GroupControll
{

    public partial class Form4 : Form
    {
        Random ran = new Random();
        int N = 0; double R = 0; //число роботов и их радиус видимости
        List<Robo> RB = new List<Robo>(); //массив с координатами роботов

        public int GetNumber //возвращает в основную форму число роботов
        {   get { return N; }  }

        public double GetRadius //возвращает в основную форму радиус
видимости роботов
        {   get { return R; }  }

        public List<Robo> GetRobots //возвращает в основную форму список
с местоположением роботов
        { get { return RB; } }

        public Form4()
        {
            InitializeComponent();
        }

        private void NumbertextBox_TextChanged(object sender, EventArgs
e)
        { if (NumbertextBox.Text.Length != 0) N =
Convert.ToInt32(NumbertextBox.Text);    }

        private void RadiustextBox_TextChanged(object sender, EventArgs
e)
        { if (RadiustextBox.Text.Length != 0) R =
Convert.ToDouble(RadiustextBox.Text);   }

        private void Addbutton_Click(object sender, EventArgs e)
//расположение роботов на плоскости
        {
            Form1 main = this.Owner as Form1;

            Graphics g = main.pictureBox1.CreateGraphics();
            Pen myPen = new Pen(Color.LightGray, 1F);

            for (int i = 0; i < N; i++)
```

```
            {
                Robo A = new Robo();
                bool check = false;
                while (!check)
                {
                    check = true;
                    A.X = ran.NextDouble() * 970;
                    A.Y = ran.NextDouble() * 680;
                    for (int j = 0; j < main.OBS.Count; j++)
                        if (Form1.Orient(main.OBS[j], A) == 0)
                        { check = false; break; }
                }
                RB.Add(A);
            }

            for (int i = 0; i < N; i++)
            {
                Form1.DrawRB(RB[i], main.OBS, R, main.pictureBox1, 1);
            }

            Close();
        }
    }
}
```